Hindawi Publishing Corporation
International Journal of Vehicular Technology
Volume 2012, Article ID 532568, 8 pages
doi:10.1155/2012/532568

Research Article

StopWatcher: A Mobile Application to Improve Stop Sign Awareness for Driving Safety

Carl Tucker, Rachel Tucker, and Jun Zheng

Department of Computer Science and Engineering, New Mexico Institute of Mining and Technology, Socorro, NM 87801, USA

Correspondence should be addressed to Jun Zheng, junzheng@ieee.org

Received 30 June 2012; Accepted 30 November 2012

Academic Editor: Nandana Rajatheva

Stop signs are the primary form of traffic control in the United States. However, they have a tendency to be much less effective than other forms of traffic control like traffic lights. This is due to their smaller size, lack of lighting, and the fact that they may become visually obscured from the road. In this paper, we offer a solution to this problem in the form of a mobile application implemented in the Android platform: StopWatcher. It is designed to alert a driver when they are approaching a stop sign using a voice notification system (VNS). A field test was performed in a snowy environment. The test results demonstrate that the application can detect all of the stop signs correctly, even when some of them were obstructed by the snow, which in turn greatly improves the user awareness of stop signs.

1. Introduction

Stop signs are the primary form of traffic control in the United States [1] because they are inexpensive and easy to maintain compared with other forms of traffic control like traffic lights. Approximately one third of all vehicular accidents occur in stop sign controlled intersections which seems reasonable considering that a majority of intersections use them. However, these accidents account for over 40% of all fatal vehicular accidents [1]. Research performed by the United States Department of Transportation (USDOT) has shown that the primary reason for this incredibly high number is due to driver inability or failure to see the stop sign, resulting in a collision.

This inability to see the stop sign is generally caused by visual obstructions. Some examples of visually obstructed stop signs are shown in Figure 1. These obstructions may be caused by plant overgrowth, frost, graffiti, parked cars, hills, and many other reasons [2–4]. For example, there is a particular intersection in Santa Fe, New Mexico, where it was reported that roughly half of all approaching vehicles missed the stop sign and continued through the intersection [5]. It was determined that the cause of such a high frequency was because of poor placement of the stop sign, which made it difficult to see from the road. If the primary reason that drivers accidentally run stop signs is because they simply do not see them, a simple solution is to use another method to convey the presence of an upcoming stop sign.

In this paper, we present a novel solution, StopWatcher, for safe drivers who want to be sure that they do not accidentally miss a stop sign. StopWatcher is a mobile application designed for the Android platform. It uses GPS, compass, and web capabilities which are available in a majority of smartphones. The primary design goal is to warn the user about upcoming stop signs on the road in real-time to improve the driving safety.

The rest of this paper is organized as follows. In Section 2, we provide a brief overview of the related work. The design of StopWatcher application is presented in Section 3. In Section 4, we describe the field test and show the test results. Finally, we conclude this paper and point out our future works in Section 5.

2. Related Work

Automatic traffic sign detection and recognition has been a popular research topic in the field of applied computer vision. It uses the traffic scene images acquired by the imaging devices mounted in the car and then applies image processing and pattern recognition techniques to detect

Figure 1: Examples of visually obstructed stop signs.

traffic signs. For example, different color information such as the hue, saturation, and HSI color space [6–8], color ranges [9], look-up tables [10] were used to segment the traffic signs in images acquired under various lighting conditions. Shape information of different traffic signs were also used in various works [6, 7, 10, 11]. However, due to the challenging factors of outdoor environment such as various illumination and weather conditions, different view angles, and so forth, those proposed traffic sign detection algorithms and systems cannot achieve perfect performance. Especially when signs are clustered or partially obstructed, the performance is generally very poor. These problems can be easily solved with our StopWatcher application as it is designed to be robust to those challenging environment factors.

Today's smartphones are equipped with numerous sensors such as GPS, accelerometers, compass, microphones and camera, and so forth, which make them useful in some innovative automobile applications. In [12], a system called Nericell was developed by Microsoft which uses accelerometer, microphone, GSM radio, and/or GPS equipped in smartphones to detect road and traffic conditions such as potholes, bumps, braking, and honking. Dai et al. utilized the accelerometer and orientation sensors in a smartphone to detect drunk driving for early alert or calling the police for help [13]. Fazeen et al. proposed to use the accelerometer and GPS of an Android-based smartphone to analyze the driver's behavior and road conditions [14]. The aim of the application is to identify road anomalies and sudden driving maneuvers for driving safety. StopWatcher has a different design goal than these applications to concentrate on automatic traffic sign detection.

3. StopWatcher: A Mobile Application for Stop Sign Detection

StopWatcher is a real-time Android application developed to make drivers more aware of stop signs, even if they are hidden, obscured, or damaged. It is also helpful when general visibility conditions are poor due to darkness or inclement weather. It uses the GPS and Compass features provided by many smartphones. The system architecture of the StopWatcher application is shown in Figure 2. The application has a sensor manager that retrieves the sensor data from the hardware and makes it available to the application. The user's current GPS data is used to retrieve the appropriate map from Google Maps for visual interpretation by the user. The application makes use of a MySQL database to store and retrieve information about intersections with stop signs. The MySQL Query Manager coordinates the retrieval of nearby stop sign data when given a user's GPS coordinates. The application then uses an Approach Classifier to interpret the sensor data, as well as the sign location data, in order to determine whether or not a user is approaching a stop sign. The programming of the application was done with Java, the programming language for Android application development.

3.1. Google Maps. While StopWatcher does not technically require any mapping software for its core functionality, we decided that it would be a nice feature for the user, and the debuggers, to be able to see which roads have stop signs ahead of time when planning a route. In this way the driver could be more prepared when actually driving to their destination. Note that the map feature is not intended to be used while the user is driving and might be disabled when the user is driving in future versions.

We use the Google Maps package in Android to handle the maps and road matching functions because it is free and has a very user friendly API. When StopWatcher is launched, it accesses the GPS on the phone and passes the coordinate data to Google Maps. It then queries the MySQL database for the positions of local stop signs. Finally it draws the standard map and centers it on the user's location. In this step it also overlays custom icons which depict the stop signs and the directions that they are facing (Figure 3). Unlike many applications that utilize Google Maps, we have forced the zoom setting to stay at 18 because we found that this setting provides the best view of all of the local streets. Other settings may cause some of the streets to disappear and the overlaid icons lose their meaning or may be misinterpreted as being on an adjacent street. Note that this would only affect the visual functionality of the program and would not affect the VNS since it is controlled by the GPS coordinate data.

There are other minor issues associated with the visual implementation of StopWatcher. While most city roads run either North-to-South or East-to-West, some do not. Since the current system uses a static set of images to denote the road directions that are required to stop at a stop sign (see Figure 3), the system could provide data that might be misinterpreted. For example, take an intersection where

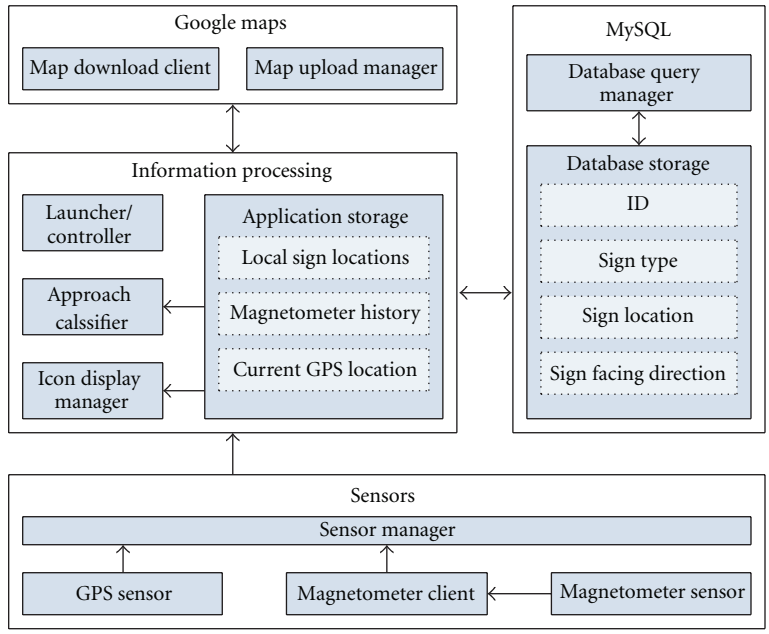

Figure 2: System architecture of StopWatcher application.

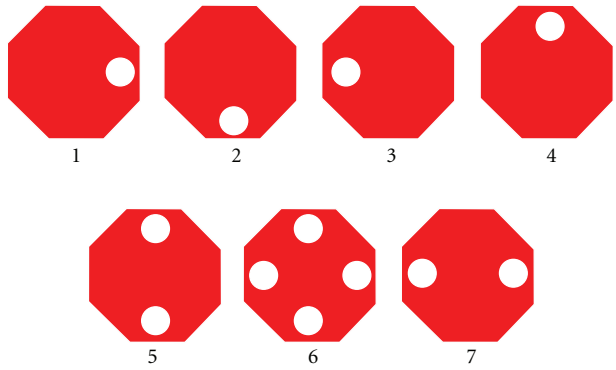

Figure 3: Symbols that StopWatcher places in Google Maps denote the directions that are required to stop at an intersection. The white dots indicate the rough directions that the stop signs are facing. For example, symbol 6 represents an all-way stop, while symbol 5 represents a North and South only stop at a 3- or 4-way intersection.

the roads run SouthWest-to-NorthEast and NorthWest-to-SouthEast and assume it contains a 2-way stop. The current icon set could cause confusion as the white dots are only on the North, South, East, and West sections of the image. We tested another method involving the use of the Google Maps snap-to-road functions. We found that in most cases it was not able to accurately place the icons due to the fact that they were being placed on either side of an intersection of two streets. This was only further compounded by roads that had curves in them, causing them to hug another street before making a sharp turn to form an intersection with it. Another problem with the snap-to-road functions is that it does not work well in small towns. This is due to Google making less effort to accurately map these places. As such we decided to stay with the current method since most streets do in fact run in a standard grid pattern and the graphical output is an accessory function, not a core function of the program.

3.2. MySQL. StopWatcher requires a database in order to hold the GPS positions of stop signs. We decided to use MySQL because it is easy to implement and effective for our proof of concept trials. Our design goal is to keep processing time and memory usage to a minimum. This is because the application is designed to be run on a smartphone which has very limited resources. In keeping with this design goal, we decided to not have entries for every stop sign, but rather have entries that contained all of the information about a given intersection. This greatly reduces the amount of storage space required for intersections containing more than one stop sign. This method offers considerable improvements to the efficiency of the application because most intersections in a city are 4-way intersections that require at least 2 stop signs to be effective [15]. A 2-way intersection occurs when a minor street intersects with a major street, while a one-way intersection only occurs when a street terminates at an intersection, which is usually located at the outer edge of the street grid. This method also reduces the number of necessary distance and approach vector computations, since there are far fewer local nodes than if we had made an entry for every sign.

Each tuple in the database table contains the following intersection data: *ID*, latitude and longitude coordinates, the direction the signs are facing, and the type of road sign at theintersection. The storage cost for each element of the tuple is show in Table 1 based on the specification of MySQL [16]. The latitude and longitude coordinates are taken from the center of the intersection and extend to six places past the decimal in order to ensure a high level of accuracy. This

Table 1: Storage costs for the elements of an entry in MySQL database.

Field name	Field type	Storage size (bytes)
ID	Integer	4
Latitude	Single-precision float	4
Longitude	Single-precision float	4
Sign facing direction	Enum	1
Sign type	Enum	1

is important because the distance across an intersection is typically around 36 feet in most cities of the United States [17]. The program compensates for this position by adding 18 feet, half the distance across a standard intersection, to the distance calculations. Note that less accurate positioning data could place the sign off the center of the intersection and cause StopWatcher to calculate inaccurate approach vectors and distances.

The "facing" field of the table is an enumerated type containing all unique combinations of "N," "S," "E," and "W" denoting North, South, East and West respectively. The 2-way stop sign combinations of NE, NW, SE and SW were excluded as they do not exist in normal traffic conditions due to their inherent ineffectiveness. The values in this field are additive, that is, an intersection with the "facing" field entry of "NS" means that the North and South streets contain a stop sign. It should be noted that the directional terms, like North, do not necessarily mean that the road at the intersection truly runs North, but rather that it is in a more Northern position than the other streets at the intersection. In the case of an intersection that runs in a diagonal to the cardinal directions, we use a 45 degree right turn to determine assignment. That is to say the street that is in the North-East would be considered North and the street in the North-West would be considered West. It should be noted that these types of street intersections were not present in our field test which our simple calculating method worked well. In our future work, we may allow for various combinations of road placements including diagonal positioning for non-standard roads and use the center of the intersection to determine the angle at which the stop signs were placed, but the computational complexity must be considered as it will run on the resource-constrained smartphone.

The last field which denotes the sign type is meant for future expansions to include other traffic signs such as yield signs, crosswalks, school zones, and so forth that could also become obstructed or are otherwise hard to see. Currently the only value being used is the Stop Sign which is the default entry.

3.3. Workflow of StopWatcher Application. Figure 4 shows the workflow of StopWatcher application. After the startup of StopWatcher, the program places the stop signs on the map as shown in Figure 5. The stop sign intersections are retrieved from the MySQL database, grouping them by their "facing" values. All stop sign intersections are then displayed as individual icons and overlaid on the Google map.

Next, the program enters the "Update" stage. This is where the phone samples the user's location via the GPS functions of the phone and detect if the vehicle is approaching a stop sign. The application updates the location every time the user's location changes by at least one meter. The main reason that we decided to force the program to update only on startup and location change is because there is no need to recalculate approach and distance vectors when the user is not moving. This helps reduce power consumption and lighten the workload of the processor when waiting at an intersection or any other time the car is not moving. On the other hand, when the user is moving, it is necessary for the update to be triggered every meter in order to keep the user's location as accurate as possible. Note that in practice the calculating of movement is based on the accuracy of GPS positioning which can vary by several meters per sample, especially when the phone is sitting still. This variance causes the phone to get a false notification to recalculate the values and renders the feature ineffective in real life use. However, should GPS calculations become more accurate, this feature would indeed help to conserve power consumption.

In addition to sampling GPS data in the update stage, the phone also samples what direction the user is currently facing. It does this by accessing the compass sensor, a magnetometer that determines which direction the phone is facing based on the magnetic pull of the Earth's poles. This bearing information is used to determine the relative direction that the user is traveling: North, South, East, or West.

The sampled GPS data and direction information will be used to determine if the user is approaching a stop sign. There are two primary states for this purpose: "Approaching" and "Searching". The approaching state denotes that the user is within range of a specific stop sign, and issues distance alerts via the VNS. The searching state denotes that the user is not currently within a stop sign's range, and it actively compares the stop sign intersection lists against current location and direction data to see if the user is approaching a stop sign. The searching state is the state that the phone starts in when the application is first activated. When it finds a stop sign intersection within range, it decides whether or not it should switch to the approaching state. This is necessary because there is the chance that what it actually found is what we call a "false approach".

In the searching state, the program looks at the stop sign intersection locations to see if the user has entered the range of a stop sign. The program does not need to search all intersection entries, because not all stop signs are relevant to the direction the user is traveling. For example, if a user is traveling North bound and they are approaching an intersection that only has stop signs on the East and West sides, then those signs are not relevant to the driver. Therefore, the program does a preliminary paring down of the stop signs to search based on their "facing" values. The program only searches through the lists of stop sign intersections that are relevant to the driver's current route.

While in the searching state, the program searches all of the relevant stop sign intersection lists, checking if the user

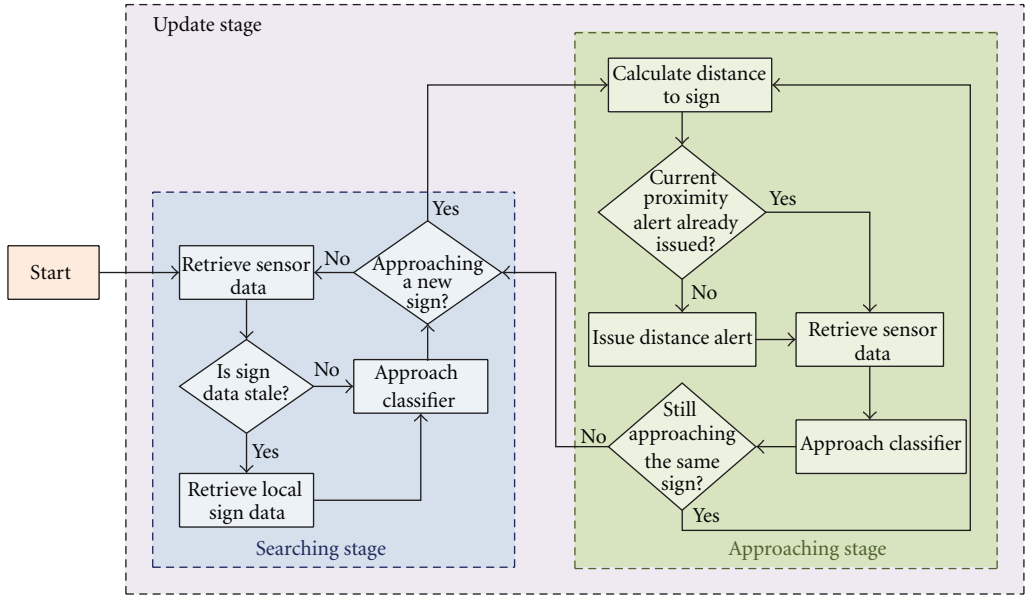

Figure 4: Workflow of StopWatcher application.

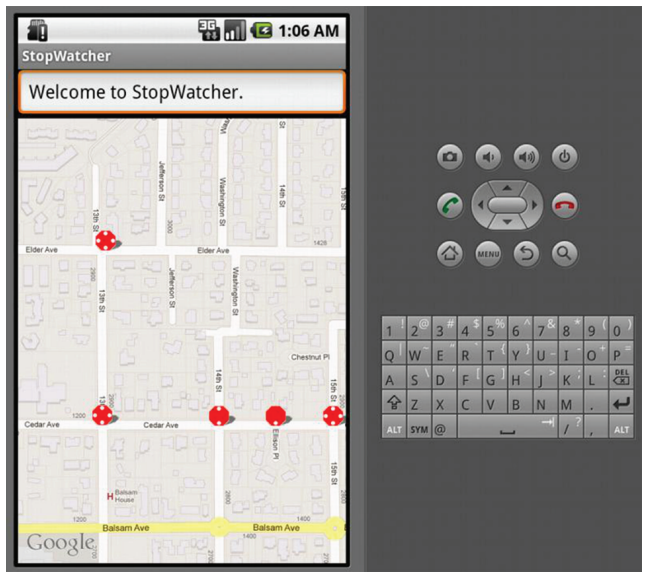

Figure 5: Screenshot of StopWatcher application's user interface as seen from the Eclipse Android SDK emulator.

has entered the range of a stop sign. We use a maximum range of 300 feet to determine when the user is considered "on approach" with the stop sign. Using omni-directional range detection can cause errors in the application. This is because a driver could be within 300 feet of a stop sign, but be on a separate road that is running parallel to the one which has the stop sign. To alleviate this type of false detection, we compare the approach angle of the vehicle to the intersection. If the angle between the user's location and the intersection's location is greater than 45 degrees off of the user's traveling direction, then the program does not enter the approaching state. This method shrinks our search area to a cone, searching 300 feet in front of the user, in a 90 degree arc as shown in Figure 6. If the program searches all relevant stop signs and does not enter the approaching state, the program concludes that the user is not approaching a stop sign, and waits for the next location update. On the other hand, if the program detects a stop sign within range that the user is actually approaching, the program enters the approaching state.

It should be noted that we assume that the user will not enter the same stop sign intersection's range twice in a row. That is to say that the driver will not re-enter the same intersection right after he or she just goes through it. This is necessary in order to prevent false detections due to minor inaccuracies and changes in GPS location data.

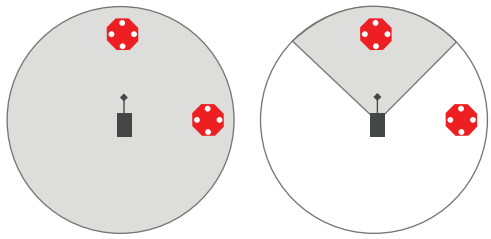

Figure 6: The program has a 300 foot search radius when looking for signs. (a) Two separate signs within range that could result in a false positive. (b) Refine the search to only include the intersections that are in line with the current direction of travel.

The approaching state is designed to issue distance alerts, via the VNS, as a user approaches a specific stop sign. There are six possible range alerts that the user may receive: they occur when the user is within 300, 250, 200, 150, 100, or 50 feet from a stop sign. However, if all of these alerts are always triggered, it will be very annoying. Therefore, at a maximum, only four alerts per sign can be triggered in the application. The alerts are designed to skip every other sound byte; that is, if the user hears the 300 foot warning, they will not hear the 250 foot warning, but will hear the 200 foot one. This is to reduce the annoyance factor caused by repeated updates, while still providing the user with the necessary information. The 50 foot warning will always sound no matter the distance that the warnings start at. This is to warn the user that they should be stopping now and allows stop sign detection even in short stretches of road. Once the last alert (50 feet) has been triggered, the program stores the approaching stop sign in a buffer, and returns to the searching state.

One issue to overcome is the scenario when the user is approaching a stop sign, but turns off the road before entering the stop sign intersection. To address this problem, two safeguards are created that continue to check the distance to the intersection and the traveling direction. The first safeguard is that if the user's distance to the stop sign intersection becomes greater than 300 feet, the user has left the stop sign's range. When this happens, the program returns to the searching state. The second safeguard checks the direction the user is traveling while approaching the intersection. If the user's direction is different than the direction of the stop sign approach three rimes in a row, the user has changed direction. There is one other requirement for this that is necessary for it to work correctly. The new direction must be consistent across all 3 samples. This is because there is always the possibility of the phone being bumped or affected in such a way as to give an inaccurate reading. Having three consistent samples helps remove almost all errors that could be caused by the phone mistakenly believing that the user has turned onto a different road. If these criteria are met, then the program will return to the searching state.

4. Field Tests

The StopWatcher application was designed and installed in a Samsung SPH-M920 smartphone equipped with Andriod OS, GPS, compass, and web capabilities. The field tests were designed to test the StopWatcher application while driving around a typical neighborhood in North America. It should be noted that the test was conducted after a small snow storm which made lots of the stop signs visually obstructed. For the test to be a success it must accurately warn the driver of all intersections where he must stop and not give false warnings when approaching an intersection where the user is not required to stop. It must also give accurate distance estimates to the intersection containing the stop sign when the user is required to stop. We determined that the application must warn the driver of actual stop signs 100% of the time to be considered a success. In addition we decided that it would be allowable to have up to 10% false positives as long as all of the actual stop sign intersections were reported. Figure 7 shows the map of the intersections that were included in the field tests.

During the field test, we set out with the goal to pass through 100 total stop sign intersections, where 50 would require stopping at the sign and the other 50 would not. During the test, all 50 stop sign warnings were triggered correctly which meets our goal of 100% accuracy for hits. However, we did encounter 4 false positive that occurred when turning North from a 4-way stop, which was the only intersection that gave us false positives. However, even with this one intersection creating false positives, we still came out with only 4 false positives out of 50 passes equaling to an 8% false positive rate which is acceptable with our 10% bar. The reason that this 4-way stop sign consistently gave false positives will be in our future investigation.

5. Discussions and Conclusion

Our conclusion is that the primary design goal of StopWatcher has been accomplished based on the preliminary results of our field test. With StopWatcher, a driver can be aware of stop signs that may be obscured or otherwise hard to see from the road. As a secondary function it helps to remove the temptation to send text messages or talk on the phone while driving. These in turn, allow drivers to be safer and more responsible when they are behind the wheel.

There are still a few areas of StopWatcher that are in dire need of improving. One important issue is the power consumption. With the StopWatcher application running, the phone only had about 30–45 minutes of battery life due

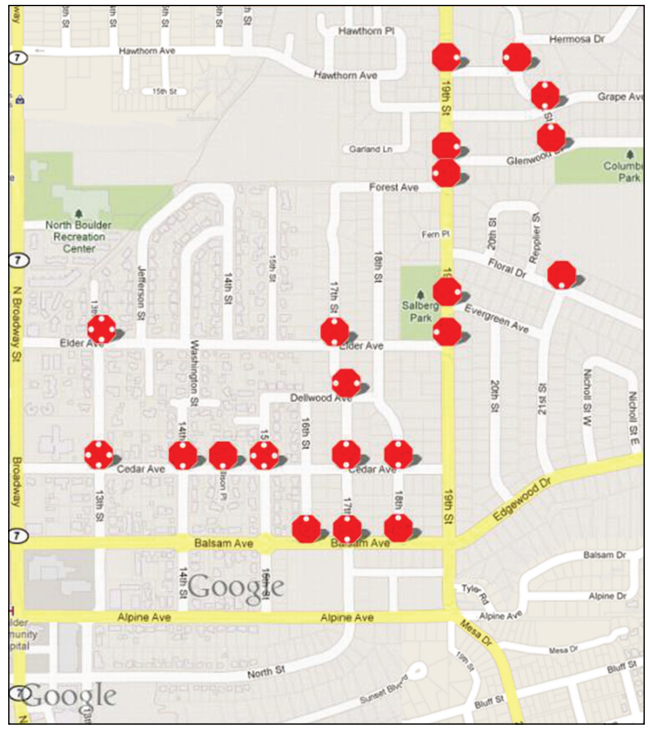

Figure 7: Map of all intersections included in the field test.

to the persistent use of GPS, compass, VNS, and graphical user interface. This may limit the usability of our application but can be mitigated by plugging the phone into the power source available in every car. Another improvement we would like to make on StopWatcher in the future is to track more than just stop signs, which has been included in our design but not implemented. In addition, we would like to implement a time system as well for traffic signs that only apply during certain times of the day. These include school zone speed limits, which only apply for about an hour before and after the school session, as well as certain parking signs. This could help people avoid putting children at risk as well as avoid some hefty traffic fines, which are sometimes doubled in school zones.

We would also like to find a way to automatically update the road sign database from the city archives. This could be a bit more problematic as most city governments do not include the exact GPS coordinates of each stop sign that is present in their city. Another possibility would be to parse street names and intersections and compare them to these stop sign maps and find the GPS data from Google Maps directly in order to populate the database.

From the view of proof of concept, StopWatcher is a successful application for driving safety awareness. But there is still a lot of room for expansion in the field of safety through environmental awareness applications. Currently, the majority of applications of location based services are focused on entertainment purposes, such as finding the closest restaurant or night club. Very few works target to improve the safety and well being of the users and those around them. We expect that more applications like StopWatcher and those mentioned in Section 2 will be developed in future to help reduce automotive accidents and other problems that could be prevented if the user knows more about what is going on around them.

References

[1] U.S. Department of Transportation Federal Highway Administration, "Low-cost safety improvements can improve safety at stop sign controlled intersections," 2002, http://safety.fhwa.dot.gov/intersection/resources/casestudies/fhwasa09010/stop_article.cfm.

[2] Visiting My Councilor, 2008, http://harrumpher.com/?cat=6.

[3] Hidden snow stop sign, 2011, http://peicanada.com/snow_bound_west_prince_pei/image/hidden_snow_stop_sign.

[4] J. Curiel, "Mill Valley: Stop sign obscured by foliage," 2009, http://articles.sfgate.com/2009-07-08/bay-area/17216162_1_sign-stop-intersection.

[5] J. Sena, "Frequently ignored stop sign leads to close calls," 2008, http://www.santafenewmexican.com/Local%20News/Downtown-dilemma.

[6] S. Xu, "Robust traffic sign shape recognition using geometric matching," *IET Intelligent Transport Systems*, vol. 3, no. 1, pp. 10–18, 2009.

[7] A. De la Escalera, J. M. Armingol, and M. Mata, "Traffic sign recognition and analysis for intelligent vehicles," *Image and Vision Computing*, vol. 21, no. 3, pp. 247–258, 2003.

[8] C. Y. Fang, C. S. Fuh, P. S. Yen, S. Cherng, and S. W. Chen, "An automatic road sign recognition system based on a computational model of human recognition processing," *Computer Vision and Image Understanding*, vol. 96, no. 2, pp. 237–268, 2004.

[9] X. Gao, L. Podladchikova, D. Shaposhnikov, K. Hong, and N. Shevtsova, "Recognition of traffic signs based on their colour and shape features extracted using human vision models," *Journal of Visual Communication and Image Representation*, vol. 17, no. 4, pp. 675–685, 2006.

[10] S. H. Hsu and C. L. Huang, "Road sign detection and recognition using matching pursuit method," *Image and Vision Computing*, vol. 19, no. 3, pp. 119–129, 2001.

[11] P. Jimenez, S. Arroyo, H. Moreno, F. Ferreras, and S. Bascon, "Traffic sign shape classification evaluation II: FFT applied to the signature of blobs," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 607–612, Las Vegas, Nev, USA, June 2005.

[12] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*, pp. 323–336, Raleigh, NC, USA, November 2008.

[13] J. Dai, J. Teng, X. Bai, Z. Shen, and D. Xuan, "Mobile phone based drunk driving detection," in *Proceedings of the 4th International Conference on Pervasive Computing Technologies for Healthcare*, pp. 1–8, March 2010.

[14] M. Fazeen, B. Gozick, R. Dantu, M. Bhukhiya, and M. C. Gonzalez, "Safe driving using mobile phones," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1462–1468, 2012.

[15] "Manual on uniform traffic control devices (MUTCD)," 2003, http://mutcd.fhwa.dot.gov/pdfs/2003/pdf-index.htm.

[16] http://dev.mysql.com/doc/refman/5.1/en/data-types.html.

[17] E. Ben-Joseph, "Residential street standards & neighborhood traffic control: a survey of cities' practices and public officials' attitudes," Institute of Urban and Regional Planning, University of California at Berkeley, 1995, http://web.mit.edu/ebj/www/Official%20final.pdf.